**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Курсовой проект**

Тема: Каркасная визуализация порции поверхности

заданного типа

Студент: Шиляева Н. С.

Группа: 80-304

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Составить и отладить программу, обеспечивающую каркасную визуализацию порции поверхности заданного типа. Исходные данные готовятся самостоятельно и вводятся из файла или в панели ввода данных. Должна быть обеспечена возможность тестирования программы на различных наборах исходных данных. Программа должна обеспечивать выполнение аффинных преобразований для заданной порции поверхности, а также возможность управлять количеством изображаемых параметрических линий. Для визуализации параметрических линий поверхности разрешается использовать только функции отрисовки отрезков в экранных координатах.

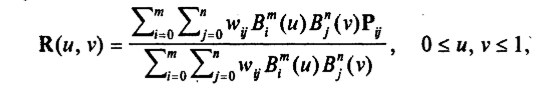
Вариант: Бикубическая поверхность Безье.

1. Решение задачи

Язык программирования: C#.

Выбранный язык удобен для решения задачи, так как он обладает удобными библиотеками для построения фигур и вывода их на экран.

Рациональной поверхностью Безье называется совокупность точек, определяющаяся по заданному массиву уравнением вида:

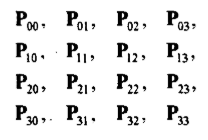




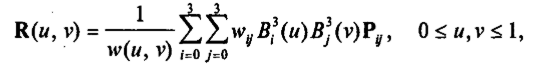
где и – многочлены Бернштейна.

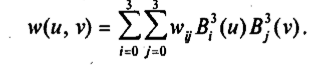
Неотрицательные числа называются весами. В случае, если все веса равны между собой, получается стандартная элементарная поверхность Безье.

*Частный случай.* При m=n=3 имеем элементарную бикубическую рациональную поверхность Безье, порожденную массивом из 16 вершин:



Такая поверхность описывается уравнением:

****

где 

Аффинные преобразования

Аффинное преобразование – комбинация линейных преобразований, результатом которых является взаимно однозначное точечное отображение плоскости или пространства на себя, при котором:

· прямые и плоскости переходят в прямые и плоскости

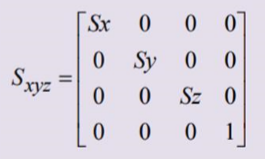
· пересекающиеся прямые и плоскости - в пересекающиеся

· параллельные прямые и плоскости – в параллельные

В курсовой работе приведены данные аффинные преобразования:

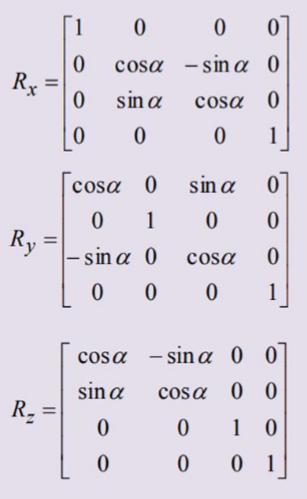
1. Масштабирование

Матрица преобразования:



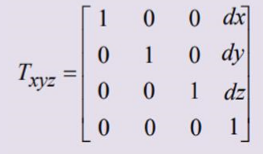
2. Вращение относительно осей

Матрица преобразования:



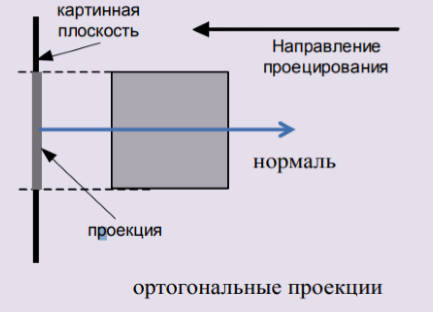
3. Перенос по осям

Матрица преобразования:



Проецирование

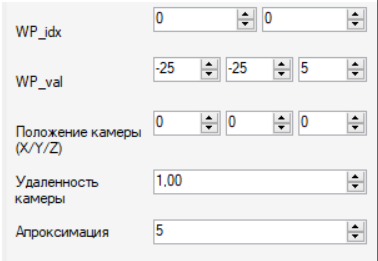
Проекции преобразуют точки, заданные в СК размерности n, в СК меньшей размерности. В курсовой работе я пользовалась ортогональным проецированием. При данном подходе направление проецирования коллинеарно с нормалью к проекционной плоскости.

****

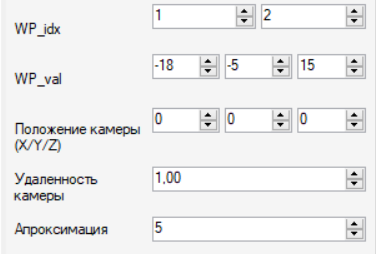
1. Набор тестов

Задаем все необходимые данные бикубической поверхности Безье, определяя необходимую вершину WP\_idx (двигаясь по указателям: вверх, вниз) и указывая координаты выбранной вершины WP\_val (вправо/влево; вверх/вниз; вперед/назад), вводим необходимое (удобное) для просмотра значение удаленности камеры и аппроксимации:

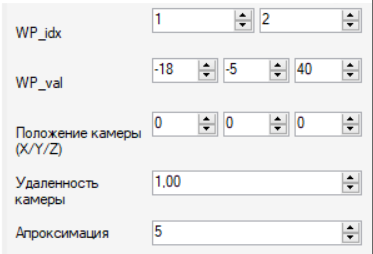
Тест 1:



Тест 2:



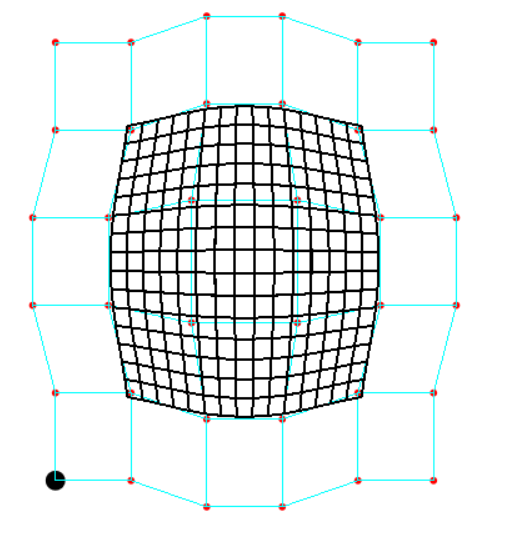
Тест 3:



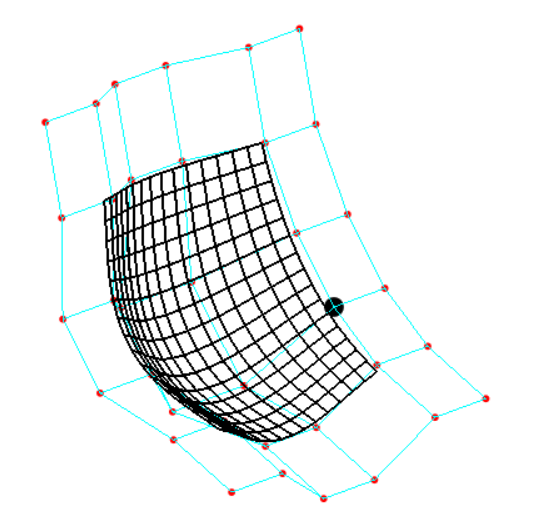
1. Результаты выполнения тестов

Наглядно видим выполнение программы по тестам, описанным ранее.

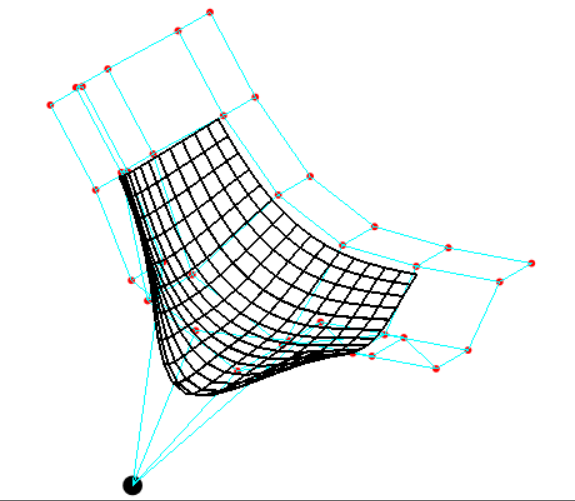
Тест 1:



Тест 2:



Тест 3:



1. Листинг программы

*Файл Program.cs*

using System;

using System.Collections;

using System.Linq;

using SharpGL;

using CGLabPlatform;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

public abstract class CGLabDemoOGL : OGLApplicationTemplate<CGLabDemoOGL>

{

[STAThread] static void Main() { RunApplication(); }

public double[,,] W =

{

{ {-25, -25, 5 }, {-25, -15, 10}, {-28, -5, 15}, {-28, 5, 15}, {-25, 15, 10}, {-25, 25, 5} },

{ {-15, -25, 5 }, {-15, -15, 10}, {-18, -5, 15}, {-18, 5, 15}, {-15, 15, 10}, {-15, 25, 5} },

{ {-5, -28, 10 }, {-5, -18, 15}, {-7, -7, 25}, {-7, 7, 25}, {-5, 18, 15}, {-5, 28, 10} },

{ {5, -28, 10 }, {5, -18, 15}, {7, -7, 25}, {7, 7, 25}, {5, 18, 15}, {5, 28, 10} },

{ {15, -25, 5 }, {15, -15, 10}, {18, -5, 15}, {18, 5, 15}, {15, 15, 10}, {15, 25, 5} },

{ {25, -25, 5 }, {25, -15, 10}, {28, -5, 15}, {28, 5, 15}, {25, 15, 10}, {25, 25, 5} }

};

public int mx = 6, my = 6;

public double dt = 0.2;

public double angleX = 0;

public double angleY = 0;

public double angleZ = 0;

public double shiftX = 0;

public double shiftY = 0;

public double shiftZ = 0;

public double toRad = Math.PI / 180;

public double xyz(int i, int j, double u, double v, int k)

{

double u0 = u - i;

double u02 = u0 \* u0;

double u03 = u02 \* u0;

double u1 = 1 - u0;

double u12 = u1 \* u1;

double u13 = u12 \* u1;

double v0 = v - j;

double v02 = v0 \* v0;

double v03 = v02 \* v0;

double v1 = 1 - v0;

double v12 = v1 \* v1;

double v13 = v12 \* v1;

double[] SV = { 0, 0, 0, 0 };

for (int i2 = i; i2 < i + 4; i2++)

return res;

}

#region Свойства

[DisplayNumericProperty(new[] { 0d, 0d }, 1, 0, "WP\_idx", 0, 5)]

public virtual DVector2 WP\_idx

{

set { if (Set<DVector2>(value)) {

int x = (int)value.X;

int y = (int)value.Y;

}

}

get { return Get<DVector2>(); }

}

[DisplayNumericProperty(new[] { -25d, -25d, 5d }, 1, "WP\_val")]

public virtual DVector3 WP\_val

{

set

{

{

Set<DVector3>(value);

W[(int)WP\_idx.X, (int)WP\_idx.Y, 0] = value.X;

W[(int)WP\_idx.X, (int)WP\_idx.Y, 1] = value.Y;

W[(int)WP\_idx.X, (int)WP\_idx.Y, 2] = value.Z;

}

}

get { return Get<DVector3>(); }

}

[DisplayNumericProperty(new[] { 0d, 0d, 0d }, 1, 0, "Положение камеры (X/Y/Z)")]

public virtual DVector3 cameraAngle

{

get { return Get<DVector3>(); }

set { if (Set(value)) UpdateModelViewMatrix(); }

}

[DisplayNumericProperty(1.0d, 0.1, 2, "Удаленность камеры")]

public virtual double cameraDistance

{

get { return Get<double>(); }

set { if (Set(value)) UpdateModelViewMatrix(); }

}

[DisplayNumericProperty(5, 1, "Апроксимация", 2, 10)]

public virtual double apr { set; get; }

#endregion

protected override void OnMainWindowLoad(object sender, EventArgs args)

{

base.VSPanelWidth = 300;

base.ValueStorage.RightColWidth = 60;

base.RenderDevice.VSync = 1;

#region Обработчики событий мыши и клавиатуры -------------------------------------------------------

RenderDevice.MouseWheel += (s, e) => cameraDistance += e.Delta / 1000.0;

RenderDevice.MouseMoveWithRightBtnDown += (s, e) =>

{

shiftX += e.MovDeltaX \* 0.2;

shiftY += e.MovDeltaY \* 0.2;

};

RenderDevice.MouseMoveWithLeftBtnDown += (s, e) =>

{

angleX += e.MovDeltaX \* 10;

angleY += e.MovDeltaY \* 10;

angleZ += e.MovDeltaX \* 10;

};

#endregion

#region Инициализация OGL и параметров рендера -----------------------------------------------------

RenderDevice.AddScheduleTask((gl, s) =>

{

gl.Disable(OpenGL.GL\_DEPTH\_TEST);

gl.ClearColor(0, 0, 0, 0);

});

#endregion

}

private void UpdateModelViewMatrix()

{

#region Обновление объектно-видовой матрицы ---------------------------------------------------------

RenderDevice.AddScheduleTask((gl, s) =>

{

// TODO:

});

#endregion

}

protected unsafe override void OnDeviceUpdate(object s, OGLDeviceUpdateArgs e)

{

double u = 0, v = 0;

int i, j;

dt = 1 / apr;

var gl = e.gl;

gl.ClearColor(1, 1, 1, 1);

gl.Clear(OpenGL.GL\_COLOR\_BUFFER\_BIT | OpenGL.GL\_DEPTH\_BUFFER\_BIT | OpenGL.GL\_STENCIL\_BUFFER\_BIT);

gl.MatrixMode(OpenGL.GL\_PROJECTION);

gl.LoadIdentity();

gl.Ortho(-40, 40, -40, 40, -40, 40);

gl.Scale(cameraDistance, cameraDistance, cameraDistance);

gl.Translate(shiftX, shiftY, shiftZ);

gl.Rotate((float)(toRad \* angleX), (float)(toRad \* angleY), (float)(toRad \* angleZ));

gl.Rotate((float)(toRad \* angleX), (float)(toRad \* angleY), (float)(toRad \* angleZ));

gl.Enable(OpenGL.GL\_POINT\_SMOOTH);

gl.PointSize(6);

gl.Color(1f, 0, 0);

gl.Begin(OpenGL.GL\_POINTS);

for (i = 0; i < mx; i++)

for (j = 0; j < my; j++)

gl.Vertex(W[i, j, 0], W[i, j, 1], W[i, j, 2]);

gl.End();

gl.PointSize(16);

gl.Color(0, 0, 0);

gl.Begin(OpenGL.GL\_POINTS);

gl.Vertex(WP\_val.X, WP\_val.Y, WP\_val.Z);

gl.End();

gl.Color(0, 1f, 1f);

gl.LineWidth(1);

gl.PolygonMode(OpenGL.GL\_FRONT\_AND\_BACK, OpenGL.GL\_LINE);

gl.Begin(OpenGL.GL\_QUADS);

for (i = 0; i < mx - 1; i++)

{

for (j = 0; j < my - 1; j++)

{

gl.Vertex(W[i, j, 0], W[i, j, 1], W[i, j, 2]);

gl.Vertex(W[i + 1, j, 0], W[i + 1, j, 1], W[i + 1, j, 2]);

gl.Vertex(W[i + 1, j + 1, 0], W[i + 1, j + 1, 1], W[i + 1, j + 1, 2]);

gl.Vertex(W[i, j + 1, 0], W[i, j + 1, 1], W[i, j + 1, 2]);

}

}

gl.End();

gl.LineWidth(2);

gl.Color(0, 0, 0);

gl.Begin(OpenGL.GL\_QUADS);

for (i = 0; i < mx - 3; i++)

{

for (j = 0; j < my - 3; j++)

{

u = i;

while (u <= i + 1)

{

v = j;

while (v <= j + 1)

{

{

gl.Vertex(xyz(i, j, u, v, 0), xyz(i, j, u, v, 1), xyz(i, j, u, v, 2));

gl.Vertex(xyz(i, j, u + dt, v, 0), xyz(i, j, u + dt, v, 1), xyz(i, j, u + dt, v, 2));

gl.Vertex(xyz(i, j,u + dt, v + dt,0),xyz(i, j, u + dt, v+dt,1),xyz(i,j,u+dt,v+dt,2));

gl.Vertex(xyz(i, j, u, v + dt, 0), xyz(i, j, u, v + dt, 1), xyz(i, j, u, v + dt, 2));

}

v += dt;

}

u += dt;

}

}

}

gl.End();

gl.Flush();

}

}

1. Вывод

В ходе данной курсовой работы были получены основные навыки по построению бикубической поверхности Безье и созданию трёхмерных объектов, в частности – использование таких направляющих, как сплайны.

Данный курс познакомил меня с компьютерной графикой. Мне всегда было интересно, как внутри устроены игры, точнее их графическая часть. Теперь я имею представление о том, как это работает.

Список литературы

1. Шилдт, Герберт. Ш57 С# 4.0: полное руководство. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2011. — 1056 с.: ил. — Парал. тит. англ. (дата обращения: 12.12.2020).
2. Прайс. C# 7 и .NET Core. Кросс-платформенная разработка для профессионалов, 3-е издание. -М.: Питер, 2018 - 640 с. (дата обращения: 14.12.2020).